A simple universal machine of type two and a formal proof of its correctness

Florian Steinberg

As a realistic theory of computing, computable analysis is based on a Turing machine model of computation. The most popular formulation of this machine model are Weihrauch's TTE machines [Wei00]. Another computationally equivalent formulation which is preferred for complexity theory of operators is that of oracle Turing machines whose oracle is an element of Baire space [KC12]. This talk takes the second perspective and considers a machine at type level two to have access to functional input by means of queries for its values on a specified input.

Mathematically such machines can be modeled by oracle Turing machines and are the standard for introducing computability on Baire space: For functional input $\varphi \in \mathbb{N}^{\mathbb{N}}$ and discrete input $n \in \mathbb{N}$ an oracle Turing machine M either produces a return value $M^{\varphi}(n)$ or diverges. A machine M is said to be defined on oracle φ if it converges for all inputs $n \in \mathbb{N}$ and is assigned the partial operator $F_M :\subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ whose domain are the functions the machine converges on and whose value $F_M(\varphi)$ on such a φ is given by $n \mapsto M^{\varphi}(n)$. An operator on Baire space is computable if it is a restriction of some F_M.

Due to the finitistic nature of the model of computation, any computable operator is continuous with respect to the natural topology on Baire space. Moreover, it is a well known fact that any continuous operator on Baire space is computable relative to an additional oracle. This basic fact is the basis for the function space construction used in computable analysis. Furthermore, it can be understood as stating the existence of a universal type-two machine. Using some standard pairing function $\langle \cdot, \cdot \rangle$ on Baire space this statement reads as follows: there is an oracle machine U such that for any continuous operator $F :\subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ there exists a $\psi_F \in \mathbb{N}^{\mathbb{N}}$ such that $U^{\langle \psi_F, \cdot \rangle}$ computes F, i.e. for any φ from the domain of F and all $n \in \mathbb{N}$ it holds that

$$U^{\langle \psi_F, \varphi \rangle}(n) = F(\varphi)(n).$$

For a fixed machine U, a functional ψ_F as above is called an associate of the operator F. Sometimes it is called a Kleene associate, as the basic ideas behind the above go back to Kleene and Kreisel [Kle59, Kre59]. There are several ways to concretely construct universal machines. On one hand, one may rely on a universal Turing machine in type-one, i.e. use a relativized version of the universal machine from traditional computability theory. This allows to shift all computable parts of the execution of an operator away from the associate and towards the universal machine and is the construction most commonly used in computable analysis. On the other hand, one may keep most of the execution of an operator locked inside of the associate. This leads to a very simple form of the machine U.

The talk gives an explicit description of such a simple universal machine and discusses some of its advantages over more complicated ones as well as some of its drawbacks: The simplicity of the algorithm driving the machine allows a direct formulation in Coq and a fairly simple formal proof of the universality property. On the other hand, the simplicity of the universal machine leads to a loss in performance: The talk gives an argument, but not a proof, that one has to expect a quadratic overhead of executing a name of an operator in the universal machine over executing the operator itself. This universal machine is closely tied to ideas from higher-order complexity theory. The talk briefly discusses two such connections [BK02, Fér17].

Finally, the talk discusses an application: The universal machine was used to extend a basic library for computable analysis in Coq by a function space construction. The algorithm of the simple machine is directly executable inside of Coq. As a consequence, functions that have been constructed can be executed with minimal overhead. For the purpose of use for this library it was convenient to use a slightly generalized version of the universal machine. This illustrates one of the merits of proving the existence of a universal machine and similar statements formally, as the generalization came up naturally during this process. It turns out that the proof of universality does not only work for the natural numbers but can easily be lifted to more general input and output types. The assumptions needed about the types are that the input types are countable and the output types are inhabited.

References

[BK02] Samuel R. Buss and Bruce M. Kapron. Resource-bounded continuity and sequentiality for type-two functionals. *ACM Trans. Comput. Log.*, 3(3):402–417, 2002. Special issue on logic in computer science (Santa Barbara, CA, 2000).

[Fér17] Hugo Férée. Game semantics approach to higher-order complexity. *Journal of Computer and System Sciences*, 87:1 – 15, 2017.

[KC12] Akitoshi Kawamura and Stephen Cook. Complexity theory for operators in analysis. *ACM Trans. Comput. Theory*, 4(2):5:1–5:24, May 2012.

[Kle59] Stephen Cole Kleene. Countable functionals. *Constructivity in Mathematics*, pages 81 – 100, 1959.

[Kre59] G. Kreisel. Interpretation of analysis by means of functionals of finite type. *Constructivity in Mathematics*, pages 101 – 128, 1959.

[Wei00] Klaus Weihrauch. *Computable analysis*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2000.